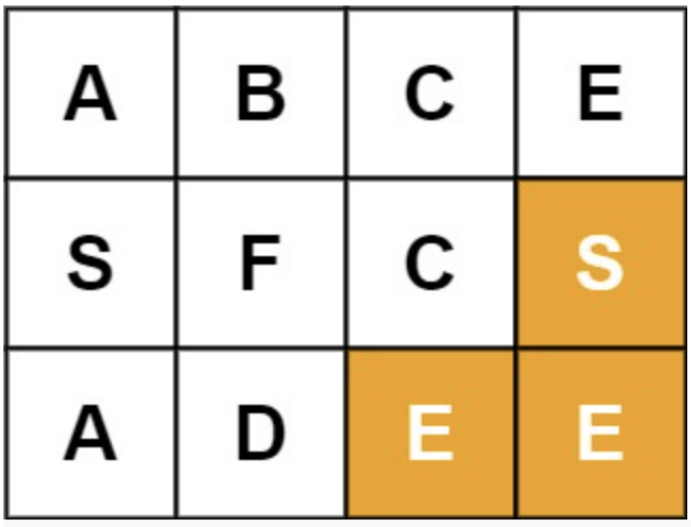
COMS 4995 Project Proposal: Word Search

Anshit Shirish Chaudhari (ac4772)
Soamya Agrawal (sa3881)

Problem Statement

Finding word in a matrix is a common game that humans love. The aim of our project is to build a parallel algorithm in Haskell to check if a word is present in the grid of characters. We say a word exists if it can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring (similar to snaking puzzle). The same letter cell cannot be used more than once. Without the loss of generality, we restrict the input grid and the query word to contain only lowercase English letters.

Example*:* In the following grid, if the user queries the existence of the word “see”, our algorithm should output true. Similarly, a query for “haskell” should output false.



Implementation

We wrote the following Python code to solve the problem sequentially. *board* is the grid and *word* is the query that the user wants to run.

```
def query_grid(self, board: List[List[str]], word: str) -> bool:
        m, n = len(board), len(board[0])

        visited = [[False]*n for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if attempt_match(i, j, m, n, board, visited, word, 0):
                    return True

        return False


def attempt_match(i, j, m, n, board, visited, word, word_index) -> bool:
    if word_index >= len(word):
        return True

    elif i < 0 or j < 0 or i >= m or j >= n:
        return False

    elif visited[i][j] or board[i][j] != word[word_index]:
        return False

    else:
        visited[i][j] = True
        resp = (
            attempt_match(i+1, j, m, n, board, visited, word, word_index+1) or
            attempt_match(i, j+1, m, n, board, visited, word, word_index+1) or
            attempt_match(i-1, j, m, n, board, visited, word, word_index+1) or
            attempt_match(i, j-1, m, n, board, visited, word, word_index+1)
        )
        visited[i][j] = False
        return resp
```

Parallelization

We see two avenues for parallelizing the above sequential algorithm.

- In the function *query_grid*, we could parallelize the calls made to *attempt_match* so that computation for each cell matching the first letter of the word will be done in parallel
- In the function *attempt_match*, we could parallelize the four calls made to itself (*attempt_match)*

We also aim to explore different parallelization strategies to address the challenge of exponential computation paths resulting from the graph based algorithm above. We will experiment with different numbers of cores and strategies to analyse the runtime on threadscope.

Stretch Goals

- Output the path in case the query word exists in the grid
- Output all the dictionary words present in the grid (like Boggle Solver)

References

- https://leetcode.com/problems/word-search/
- https://leetcode.com/problems/word-search-ii/